CORE-MATH : quand pourra-t-on calculer correctement ?

Paul Zimmermann

12èmes Rencontres Arithmétique de l'Informatique Mathématique, 27 mai 2021

Quelle bibliothèque a raison ?

```
#include <stdio.h>
#include <math.h>

int main()
{
  float x1 = 1.01027, x2 = 1.775031328;
  printf ("sinf(x1)=%.9f sinf(x2)=%.9f\n",
          sinf (x1), sinf (x2));
}
```

GNU libc 2.33 :

sinf(x1)=0.846975386 sinf(x2)=0.979216456

Intel CC 19.1.3.304 :

sinf(x1)=0.846975446 sinf(x2)=0.979216397

Quelle version a raison ?

```
#include <stdio.h>
#include <math.h>
#include <gnu/libc-version.h>

int main() {
  printf("GNU libc version: %s\n", gnu_get_libc_version ());
  double x = -0x1.f8b791cafcdefp+4;
  printf ("sin(x)=%la\n", sin (x));
}

$ gcc -fno-builtin sin.c -lm
```

GNU libc version 2.27 :

sin(x)=-0x1.073ca87470df9p-3

GNU libc version 2.33 :

sin(x)=-0x1.073ca87470dfap-3

Quel processeur a raison ?

```
#include <stdio.h>
#include <math.h>
int main()
{
  double x = 0x1.01825ca7da7e5p+0;
  printf ("x=%la y=%la\n", x, acosh (x));
}

$ icc -fno-builtin test_acosh.c # icc version 19.1.3.304
```

sirocco14.plafrim.cluster (Intel Xeon Gold 6142) :

x=0x1.01825ca7da7e5p+0 y=0x1.bc8c6186687cbp-4

zonda03.plafrim.cluster (AMD EPYC 7452, même binaire) :

x=0x1.01825ca7da7e5p+0 y=0x1.bc8c6186687cap-4

L'arrondi correct (CR)

Soit une fonction mathématique f, et un mode d'arrondi, par exemple RN (au plus proche).

Pour un flottant IEEE x, l'arrondi correct de $f(x)$, noté $\mathrm{RN}(f(x))$, est le flottant y le plus proche de $f(x)$.

IEEE 754 impose l'arrondi correct pour $x+y$, $x-y$, xy, x/y, $\sqrt{x}$, mais pas pour $\exp(x)$, $\sin(x)$, x^y, ...

IEEE 754 et le langage C

Plus précisément IEEE 754-2019 recommande (par exemple) une fonction `exp` (Section 9 : Recommended operations).

La section 9.2 dit *A conforming operation shall return results correctly rounded for the applicable rounding direction for all operands in its domain*.

Pourquoi alors la fonction `exp` du langage C ne garantit pas l'arrondi correct ?

Le standard C

Les fonctions mathématiques du langage C sont définies dans l'annexe F (*IEC 60559 floating-point arithmetic*).

L'annexe F contient une première table (*Operation binding*) donnant la correspondance entre fonctions IEEE 754 et leurs noms en C. Pour cette table, l'arrondi correct est requis.

IEC 60559 operation	C operation
addition	`+,fadd,faddl,daddl`

Une seconde table contient les fonctions mathématiques (page 444 de `n2596.pdf`) pour laquelle il est écrit : *Correct rounding, which IEC specifies for its operations, is not required for the C functions in the table*.

IEC 60559 operation	C function
exp	`exp`

Comment imposer l'arrondi correct ?

- attendre 2029 et la prochaine révision de IEEE-754, convaincre le comité de révision, et attendre que le standard C et les libm s'adaptent ;
- construire une nouvelle libm avec arrondi correct, et la maintenir pour les différents processeurs, compilateurs et systèmes d'exploitation ;
- écrire des routines efficaces avec arrondi correct, et les proposer pour inclusion dans les libm existantes (GNU libc, Intel Math Library, AMD Libm, Redhat Newlib, OpenLibm, Musl, ...)

Bibliothèques mathématiques (libm)

	`binary32`	`binary64`	`binary80`	`binary128`
GNU libc	√	√	√	√
Intel Math Library	√	√	√	√
AMD Libm	√	√		
Redhat Newlib	√	√		
OpenLibm	√	√	√	
Musl	√	√	√	

La GNU libc est disponible dans la plupart des distributions Linux.

La bibliothèque d'Intel (IML) est disponible comme image Docker.

Musl est la bibliothèque mathématique d'Alpine Linux.

IML et la majeure partie d'AMD Libm ont leur propre code, les autres bibliothèques partagent de nombreuses fonctions.

Travaux antérieurs

- MathLib/libultim (Ziv, 1991) : code inclus dans GNU libc, mais “slow path” supprimé à partir de la version 2.28, et jusqu'à la prochaine (2.34 début août). Code obscur, grosses tables, très lent sur les pires cas ;
- CRlibm (Daramy-Loirat, Defour, de Dinechin, Gallet, Gast, Lauter, Muller, 2004-2006) : code bien meilleur sur les pires cas, preuves de correction, petites tables, utilisation du FMA. Code plus maintenu, pas intégré dans les libm actuelles.

Pourquoi maintenant ?

- le prochain standard C inclura des noms réservés `cr_sin` pour des fonctions avec arrondi correct ;
- progrès récents dans la recherche des pires cas (projet TaMaDi 2010-2013, outil BacSel, thèse de Serge Torres 2016) ;
- progrès récents dans l'implantation (projet MetaLibm, 2014-2018) ;
- très bonne connaissance du domaine (Handbook of Floating-Point Arithmetic) ;
- contacts avec les développeurs des libm, avec le *C Floating-Point group* ;
- de plus en plus de demande des utilisateurs (précision et reproductibilité numérique).

Le projet CORE-MATH

Phase 1 : publier du code CR efficace pour `cbrt` et `acos`

Phase 2 : contacter les développeurs des libm et leur proposer de rejoindre le projet (observeur ou développeur). Discuter avec eux de détails techniques (licence, taille des tables, du code, ...)

Phase 3 : implanter du code CR avec preuves, et aider les développeurs des libm à l'intégrer

Phase 4 : le cas échéant, réparer les bugs dans le code et/ou les preuves

Quelles fonctions ?

Les 39 fonctions définies dans IEEE 754-2019 :

- `exp`, `expm1`, `exp2`, `exp2m1`, `exp10`, `exp10m1` ;
- `log`, `log2`, `log10`, `logp1`, `log2p1`, `log10p1` ;
- `hypot` ;
- `rSqrt` ;
- `compound` ;
- `rootn`, `pown`, `pow`, `powr` ;
- `sin`, `cos`, `tan`, `sinPi`, `cosPi`, `tanPi` ;
- `asin`, `acos`, `atan`, `asinPi`, `acosPi`, `atanPi` ;
- `atan2`, `atan2Pi` ;
- `sinh`, `cosh`, `tanh` ;
- `asinh`, `acosh`, `atanh`.

Quels formats ?

- simple précision (`binary32`) ;
- double précision (`binary64`) ;
- double précision étendue (`long double` sur x86_64) ;
- quadruple précision (`binary128`).

Comment arrondir correctement ?

Soit un format cible avec une mantisse de n bits.

Soit m une borne sur les pires cas (en général $m \approx 2n$), et $\circ()$ le mode d'arrondi.

- [fast path] calculer une approximation y avec environ $n + 10$ bits corrects, et une erreur bornée par ε
- [test d'arrondi] si $\circ(y - \varepsilon) = \circ(y + \varepsilon)$, renvoyer ce nombre
- [slow path] sinon calculer une approximation y' avec au moins m bits corrects, alors $\circ(y')$ est toujours CR

Plan pour une fonction et un format donnés

Calculer les pires cas et le cas échéant les cas exacts.

Essayer plusieurs implantations du fast path et du slow path : différentes réductions d'argument, différents formats internes (`double`, `int64_t`), utilisation du FMA. Vérifier l'arrondi correct sur les pires cas et les cas exacts.

Garder la version la plus rapide et prouver sa correction (preuve papier, avec calculs sur ordinateur si besoin, en utilisant des outils comme Sollya ou Gappa).

Publier le code et sa preuve, et aider les développeurs de libm à intégrer le code.

Pires cas

Flottants x tels que $f(x)$ est très proche d'un flottant (ou du milieu de deux flottants pour l'arrondi au plus proche).

$\exp(0.0940782231357287 8) = 1.0986456820663385000000000000000278$

Pour la double précision, ils ont été calculés pour la plupart des fonctions (travaux de Muller et Lefèvre, projet TaMaDi).

Dans certains cas (atan2/hypot/pow en simple précision, sin en double précision, toutes les fonctions en quadruple précision), il est (très) difficile de calculer les pires cas.

Alternative : à la fin du slow path, faire un 2e test d'arrondi.

S'il réussit, la valeur calculée est CR.

Sinon, elle peut être fausse. Lever une exception si besoin.

Comparaison entre GNU libc 2.27 et 2.33

Intel Core i5-4590, turbo-boost désactivé, gcc 10.2.1, double précision, arrondi au plus proche.

	GNU libc 2.27	GNU libc 2.33
exp	1679/6110/0.5	24/91/0.511
log	95/423/0.5	21/97/0.520
sin	192/584/0.5	92/305/0.516
cos	194/612/0.5	107/309/0.516
pow	128/240/0.5	35/113/0.523

Légende : $x/y/z$

x : nombre moyen de cycles

y : nombre maximal de cycles

z : erreur maximale connue en ulps

Premiers résultats

Temps en cycles (moy/max), Intel Core i5-4590, gcc 10.2.1, GNU libc configuré avec `-march=native`.

cbrt	GNU libc not CR	CORE-MATH code CR
`binary32`	69/250	27/49 [1]
`binary64`	63/251	54/79 [1,2]
`binary80`	200/218	110/144 [2]
`binary128`	2400/2520	343/386 [2]

[1] code d'Alexei Sibidanov

[2] pas encore prouvé, mais passe les cas exacts et les pires cas

Code disponible à `www.loria.fr/~zimmerma/CORE-MATH`

Comment traiter les différents modes d'arrondi ?

Une seule routine pour chaque fonction mathématique :

```
fesetround (FE_TOWARDZERO);
y = cr_sin (x);
```

C'est déjà le cas pour la fonction `sqrt` par exemple. (La fonction `sqrtf` de Newlib n'est pas CR pour les arrondis dirigés.)

On calcule en interne une approximation `h` de $\sin x$ de même format que y, et `l` la correction correspondante :

```
return h + l;
```

L'arrondi correct est assuré par l'arrondi de l'addition `h + l` avec le mode d'arrondi courant.

Comment contribuer ?

- contacts avec les libm, avec le comité de révision IEEE 754, avec le *C floating-point group*
- recherche de pires cas
- écriture de code CR efficace
- validation du code (cas exacts, pires cas, tests aléatoires) et benchmarks de son efficacité
- preuves papier (et éventuellement formelles)

Me contacter si vous êtes intéressé(e).